Aaron Austin

August 18, 2024

Project Two

The overall approach I used for creating the test environment for this project revolved around ensuring my tests were well aligned with the goals of the client. I ensured I tested for the features the client requested, nothing more nothing less. This ensures that the program runs efficiently while minimizing the testing workload. My tests were not geared toward usability testing because this project did not include a user interface so I instead focused my testing where testing was needed and would be useful.

For the contact service, I focused on ensuring the create-ability, readability, updating, and deleting (CRUD) features executed correctly and efficiently. To ensure this, I designed my tests to validate data and user inputs. I ensured that all inputs were secured and wrapped to prevent leakage and ultimately security concerns. I ensured the tests also checked that the contact information updated correctly, and the program properly retrieved the contact information by the ID feature. My test checked that the program correctly deleted a contact. The tests were designed to see what happens when all these features were met correctly and how they responded when they were not met correctly to ensure errors were correctly thrown when needed. The tests were correctly aligned with the requirements of the client because the four sections of CRUD were all tested and passed the testing. This shows that all basic forms of data interaction are securely rendering as they should, programmatically. To be more specific, my unit tests were very detailed such as if a requirement was to ensure an ID was only a certain number of characters, I added a specific case in my tests to confirm that requirement was tested and passed.

For the task service, I took a similar approach as the contact service but tweaked my approach to meet the task-related requirements. I ensured my test were designed to check that tasks were being created correctly with the required fields. I ensured the tasks could not be created using invalid inputs, such as a wrong time or date. The tests checked that the tasks were updating properly and deleting without error. I ensured the task service unit tests were aligned with the requirements by creating detailed tests that were created with cases taken from those requirements, like the contact service section. An example of this would be I created a test that ensured a task could not be completed with missing required information mandated by the requirements of the client.

The appointment service class took a similar approach to the other two classes. The tests were designed to ensure the appointment creation feature functioned as required by the specifications. To facilitate an efficient testing setup for this my tests checked that appointments were created with a variety of errors to verify how the code would respond. For example, I would create a test that was out of working hours to see that when I try to create that appointment, I get an error. I ran several tests to further ensure my thought process was in tandem with what the client was wanting. For example, for this test I really focused on edge cases such as appointments overlapping with one another.

I believe my JUnit tests were of the highest quality and accuracy, testing for a wide range of use cases. I incorporated a good amount of edge cases into my test to ensure I was getting the most holistic view of the health of the program. My coverage report indicated somewhere between a 90-95% coverage over all three services. This is evidence of the good health and high efficiency of my tests.

Coming into this project I had little to no experience with JUnit tests, but after writing several tests for this client I am a lot more confident setting up the environment and writing tests. To make sure I was technically sound I had to triple check my test and then test my test to ensure they could test my code. As an example, in the task service section, I wrote the testCreateTaskWithValidData() method to confirm a test could be created with all necessary information. I then passed data through this method to try and “break” the method to ensure all input data performed as it should. I also utilized assertions throughout my code such as assertEquals() to make sure output exceeded expectations from the client. To ensure the code was efficient I utilized setup and takedown keywords such as @Before in front of my setUp() method. This allowed me to utilize test data in multiple tests throughout my code, thus limiting redundant code.

The two techniques I used in my testing approach were unit testing and boundary testing. I used unit testing to make sure blocks of code work correctly separate from the other code. This isolates issues making them easier to find, fix, and retest. For example, in my contact service, many of those methods were tested using a unit testing approach because I tested the add, update, and delete operations separately from each other to ensure they each worked properly. I used boundary testing mostly where I needed to test edge cases such as in my appointments service to test how the cases responded to dates and times that were out of range.

I did not use integration testing or functionality testing in my project. I chose not to use integration testing because aside from the services interacting with each other they did not integrate with any outside software. It could have still been valuable to test but I did not find it necessary for this project. I did not do a system test because my focus was on unit testing which implies the project is still in the early stages of completion and system testing is best for the later stages.

Unit testing would be ideal for many kinds of development situations, but mostly for testing methods on a holistic level. It would likely work better early on when a collection of like-minded methods is being created to test them in isolation before proceeding forward to other parts of the project. Boundary testing is best used when you need to test edge cases to ensure all possible outcomes, or as close as possible, are accounted for. Integration testing is beneficial when your code or the project is needing to interact with other software or systems. You want to test that the connection between the two is stable and strong. System testing is best used in the later stages of development when you can test the totality of the project and ensure it is in good health and meets the requirements set forth for it.

Anytime to are writing code in general you should be cautious of what you are doing and how you are doing it because when you lay a foundation of bad code and then build on that foundation, once you realize you need to change the foundation you will likely have to make changes in other places that were built on top of the foundation. So, ensuring you know how all your pieces of code connect with each out is crucial. If I were to alter the logic in the appointment service, then I would then need to ensure I check the logic in any method dependent on the area I changed.

The easiest way to control bias is to stick to the requirements. All my tests were designed strictly based on the requirements, so I found no concern with any bias related issues. Bias could be a concern if I was working without requirements on my own code. For example, had I have not stuck with the requirements and went off my own desire I could’ve created test in the task creation that I thought was correct and met my standards but not the requirements from the client. Though the code may have been correct the test did not meet the expectations of the client. but to combat that I would lay out requirements before I start coding and then stick to those requirements.

Code is one profession where attention to detail is highly important. Bugs in code can be so hard to find and so time consuming that comes have created positions just for those individuals to find and report those bugs. The mistake that slipped by because my test did not meet the clients requirements or partially met the clients requirements due to a lack of thorough test cases could cost a company a fortune, it could cost a patient serious injury, such as the Therac-25 incident in 1985 where a coding error cost a number of patients to be given an overdose of radiation leading to deaths and lawsuits. Ensuring my tests are as thorough as possible can save myself a ton of headache and possibly my job or worse, so I’d rather do more now than to have to pay for it later.

Work Cited

Holzman, E. (2023, June 16). *The worst computer bugs in history: Race conditions in therac-25*. BugSnag. https://www.bugsnag.com/blog/bug-day-race-condition-therac-25/